D0

MIDAS MANUAL

3 January 1980

by

Edward R. Fiala

Xerox Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA. 94304

Filed on: [Ivy]<D0Docs>D0Midas.Press
Sources on: [Ivy]<D0Source>D0MidasManual.Dm

This manual describes a largely machine-independent loader/debugger for microprocessors originally developed for the Maxc2 computer and since used for the Dorado, D0, and M68 microprocessors. This manual is specialized for the D0 version of Midas.

TABLE OF CONTENTS

LIST OF TABLES

1. Introduction

Midas is a loader/debugger that runs on an Alto and controls its target machine remotely. It can load/dump microprograms assembled by Micro, examine and modify storage, and test hardware in several ways. Versions exist for Maxc2, Dorado, D0, and M68 microprocessors.

Midas is coded about 95% in Bcpl and 5% in assembly language. The Maxc2 version was implemented by E. R. Fiala and H. E. Sturgis. The Dorado, D0, and M68 versions consist of machine-independent modules implemented by E. Fiala (Overlay and LoadRAM packages implemented by L. Deutsch and Alto microcode by E. Taft are also used) and machine-dependent sections implemented by E. Fiala for Dorado; D. Swinehart and P. Baudelaire for M68; and D. Charnley, C. Thacker, B. Rosen, C. Hankins, and E. Fiala for D0.

An internal description of Midas is available to anyone interested in adapting Midas to a new hardware system (see [Ivy]<DoradoDocs>MidasInternal.Press).

2. Storage Requirements

Midas requires about 500 Alto disk pages, using the following files:

Midas.Run	~350 pages	
Midas.Syms	~40 pages	
Midas.Errors	~8 pages	Error message strings for Midas swat calls
Midas.Programs	~2 pages	(Discussed below)
Midas.UserPrograms	~2 pages	(Discussed below)
*.Midas	~2 pages each	Command files for "RunProg" and "Read-Cmds" actions
*.mb		Assorted micro-binary files loaded by command files
Midas.RunProg	~31 pages	Built by Midas/I
Midas.Dtach	~31 pages	Built by Midas/I
Midas.FixUps	~2 pages	Created by Midas/I (used when loading .MB files)
Midas.Compare	~2 pages	Created by Midas/I, written when "Compare" fails

D0 Midas can be obtained by loading [Ivy]<D0>D0MidasRun.Dm and retrieving <D0>Midas.Programs with Ftp. You must do Midas/I to initialize Midas on your disk after retrieving these. Subsequently, new versions of Midas can be retrieved by executing the D0NewMidas.Cm command file from the Alto Executive. Midas runs only under OS versions 17 or later.

To setup an Alto disk for use in D0 microcode development or hardware debugging, you can install the Alto OS on a blank disk using the long installation dialog and erase the disk. When this finishes, fetch [Ivy]<D0>D0DebuggingDisk.Cm and execute this command file from the Alto Executive; it will retrieve Midas and a number of other files needed when using an Alto for D0 hardware debugging and microcode development.

[Rosebay13]<D0>
D0MidasRun.dm

3. Starting and Exiting from Midas

Midas may be started from the Alto Executive in the following ways:

midas/i	initializes (required when any Midas files move or change);
midas	simply fires up Midas;
midas debug	starts Midas and immediately reads commands from the "Debug.Midas" command file

Note: You are *required* to type midas/I when you change any of the files "known" to Midas; these files include the Alto OS, Midas.Run, Midas.Programs, Midas.Midas, Midas.UserPrograms, any of the *.Midas files that appear in the "Run-Prog" or "Read-Cmds" menus, and any of the *.Mb files used by the "Run-Prog" menu items.

At present, Midas will boot its connected D0 at startup. The "Boot" action, which appears in both the main command menu and the submenu put up by "Go," will cause the D0 to load and start its boot loader (which overwrites any microprogram previously loaded); Midas then communicates with the boot loader to load its Kernel microprogram, which resides in parts of pages 0, 16, and 17 of the microstore; subsequently, interactions between Midas and the D0 are carried out by messages exchanged between the Kernel microprogram and Midas using the Alto's Diablo printer interface.

> The locations used by Kernel are available to microprogrammers through the file KernelOccupied.Mc which is in [Ivy]<D0Source>KernelSources.Dm; there are provisions for overwriting most of Kernel when running large microprograms.

In the initial display arrangement, the left-hand display column shows the principal D0 registers, and the right column shows several other items. When you initially start Midas and after a "Run-Prog" action, the display will be set to this configuration.

To exit from Midas you may type either SHIFT-SWAT (i.e., simultaneously depress the left-hand shift key and the right-most, lowest unmarked key) or ";Q" or you may bug "Exit"; this will close any open output files prior to exit. Note that on exit, if the D0 was running, it will not be disturbed.

4. Midas Display and the Mouse

The Midas display is arranged as follows:

Blank area at the top (unused);
20 lines x 3 columns of name-value menus;
Blank line;
Program and elapsed time line;
Blank line;
Two command comment lines;
Blank line;
Three lines of command menu;
Blank line;
Input text line;
Blank area a the bottom (unused).

The program line will show the Midas release date or the name of the last program loaded. The right-most part of this line will show elapsed time during long-running actions such as "Go" or "Test"; it shows the execution time of Midas initialization, the last command file, or the last action at other times.

Midas uses the two comment lines to report results of actions that it executes.

When you move the mouse over a name-value menu or the command menu, the selected item (if any) inverts black and white. Mouse actions execute when you RELEASE all mouse buttons, so you can move the mouse with buttons depressed without causing damage. If the mouse has moved off of the menu that was selected when the first button went down, nothing will happen when the buttons are released.

Some menus have additional actions "underneath" the ones normally displayed which will appear when you depress appropriate button combinations, as discused below. In other words, when you DEPRESS buttons, the menu may change; when you RELEASE ALL BUTTONS the selected action will get executed. On D0 Midas, only name-value menus have actions underneath the ones normally displayed.

Since you can neither depress a button combination simultaneously nor release the buttons simultaneously, Midas accumulates the *union* of all buttons that go down. This button-union governs the "underneath" menu displayed, if any, and is the argument passed to the action procedure when all buttons are finally released.

5. Name-Value Menus

A name-value menu may contain a *register* or *memory address* in the name area and its contents in the value area. A memory address may be specified as the memory name and word number, or as the name of an address symbol defined in a microprogram you have loaded. The address symbol may be followed by +/- displacement. If a number (default radix 8) is examined, the memory name is defaulted to "VM," so examining "1234" will cause "VM 1234" to be displayed.

Name-value areas are of *different sizes.* Smaller menus on the left are already filled in when you fire-up Midas; others are empty. Any item can be put in any menu, but larger menus on the right are better for items with long names or values. If an item overflows its menu, the right-most characters of its name get truncated, then the left-most characters of its value.

To display a new item, type its name (which will appear on the input text line), move the mouse over the name field in a name-value menu, and push-and-release the *left (top)* mouse button. Memory addresses in your microprogram may optionally be followed by a displacement "+n" or "-n"; " n" is the same as "+n". Midas will obtain the value of the item from the hardware and display it.

If the command line is empty, the selected menu will be cleared when the button is released.

The address and data items in a name-value menu are affected by the *radix* and *display mode* for the item, initially defaulted from a table indexed by the register or memory number. The address offset and value radices are always identical--Midas does not allow these to be independently specified. On D0, octal radix is the default for everything; the user may change the radix with the actions discussed below.

The display mode for a value may be either *numeric, search,* or *symbolic.*

> Numeric mode shows the value as a sequence of numbers (in the chosen radix) separated by blanks; this is the default for all items.
>
> Search mode shows the value as an address symbol plus offset; this is illegal except for registers or memories that normally contain pointers into some other memory (e.g., on D0, search mode for CIA, TPC, APC, or CALLER shows the nearest IM address symbol less-than-or-equal to the value plus an offset). Search mode is not the default for any memory or register because it is slightly slower than numeric mode due to symbol table access and because more screen area is required to accommodate long address symbols; however, you may find search mode convenient for some of the items mentioned above.
>
> Symbolic mode results in a special procedure being called to print the value for the item; special procedures do not exist for any registers or memories on D0.

When Midas thinks that the value in a register may have changed, it reads its value from the hardware and updates the display; the times when Midas does this are discussed later. Names are sometimes preceded by *, indicating that the value has changed, or by ~, indicating that Midas was unable to read the value for some reason (e.g., the machine was running). For an item marked with ~, the old value, which might be wrong, is displayed.

Once some register or memory address has been put into a name-value menu, various mouse button combinations over the name or value may be used to modify the way it is displayed, sequence through words in a memory, pretty-print the value on the comment lines, or show address equivalences. These are summarized in the table below:

Buttons	*Name-field*	*Value-field*
Left	Examine	Change value
Middle	Alternate printout	Pretty-print value on comment lines
Right	A+1, A-1 menu	Append value to input line
Left+Middle	Radix menu	Radix menu
Middle+Right	Fill column menu	Display mode menu

When a button combination selects an alternate menu, the alternate menu will replace the standard menu while the mouse buttons are depressed; if you release the buttons over an alternate menu item, it will be executed; if you are outside the menu when the buttons are released, the standard menu will be restored and nothing will happen.

The "A+1", "A-1" menu appears for memory addresses, but not for registers; these increment or decrement the memory address in the menu, displaying the successor or predecessor. The "FillC" menu allows you to examine successors (A+1, A+2, etc.) in the menus below the selected one; the whole column is filled with successors, if the input text line is blank; otherwise, the input text line is evaluated to a number N, and N lines are filled in with successors. The last address examined is left on the input text line, so you can iterate the examine and fill column actions to achieve scrolling.

Releasing the *left button* over a value stores the value of the input text (or 0 if no text typed) in the selected register. For memories and registers whose values are displayed as several fields, the input text must also be divided into fields; omitted fields are zeroed. Each field may consist of numbers or memory addresses separated by +/-; expressions are evaluated using the radix for the item.

> Note: On D0 and Dorado, IM memory words show an absolute address with each value; it is impossible to modify this address from Midas--the correspondence between virtual and absolute addresses can only be established by loading a microprogram. Several other items also have read-only fields that cannot be written, as discussed later.

Provision is made for *special input evaluation* based upon the register or memory; whenever the input text cannot be evaluated as a sequence of fields, the special input procedure (if any) is called. At the present time, special input procedures are implemented for registers and memories that contain microinstructions (IM, IMX, and MIM on D0) and for 16-bit registers. These are discussed later.

Releasing the *middle button* over a value pretty-prints the value on the command comment lines. The alternate for registers that normally hold IM addresses is the nearest IM address tag less-equal to the value+offset. Registers and memories that contain microinstructions may also be printed symbolically. Other pretty-print information is detailed later.

Releasing the *right button* over a value item appends the text of the value to the input text line. This is primarily used in command files to move values from one register to another or to display

a memory address that is pointed to by the value in some other register.

6. Command Menu

The command menu holds a list of *actions* that Midas can execute. The basic menu is modified under some conditions. For example, the "Dump" menu item only appears after you have done a "Load". During execution, some actions show alternate menus.

For all D0 actions in the command menu, mouse buttons are equivalent. Many common actions may alternatively be initated by *keyboard command characters*, as given in the action table below.

Table 1: Command Menu Actions

Input	Char	Menu Item	Comments
Actions (potentially) available on all implementations of Midas			
[File]		Run-Prog	Reset symbol table, breakpoint table, and display, then do Read-Cmds.
[File]		Read-Cmds	Executes command file (def. ext. ".Midas") on input text line or from submenu.
		Show-Cmds	Shows command file text for selected menu items.
File		Write-Cmds	Write subsequent commands on File.
Files	;L	Load	Loads .MB files (names separated by ",").
Files		LdSyms	Loads only addresses from .MB files.
[File]	;D	Dump	# Dumps compacted .MB file using the .MB file(s) of the previous load to control what's dumped.
[File]	;C	Compare	# Compares hardware data to that in .MB file.
Addr	=		Prints value of an address (illegal in com-file)
IMaddr	;B	Break	Inserts break point.
[IMaddr]	;K	UnBreak	Removes breakpoint (default address = last break).
[IMaddr]	;G	Go	* Start at address (continue if nothing typed).
	;C	Continue	* Start at address without IOReset or control section reset (continue from break if nothing typed).
	;P	Continue	* (makes ";P" and ";C" synonymous)
[IMaddr]	:	SS	* Single-step at address (continue-step if nothing typed).
[IMaddr]	;S	SS	* So ":" and ";S" are synonymous.
		Test	* Test register, memory, or other test with data pattern and item selected from submenus.
		Test-All	Test everything.
		Virtual	Changes IM address interpretation to be virtual.
		Absolute	Changes IM address interpretation to be absolute.
Actions peculiar to D0 Midas			
		Boot	Boots the D0 and resets Midas as for Run-Prog
		ClrAddedBPs	Clears all breakpoints added since the last Load.
		ClrAllBPs	Clears all breakpoints.
		ShowBPs	Show BP's added since the last Load.
	;Q	Exit	Exits cleanly from Midas--";Q" and Shift-Swat are synonymous.

* = requires preceding "TimeOut" action in command file
\# = requires confirmation with <cr>, "Y", or "." (or by preceding "Confirm" command in com-file)
[...] = optional input text

Some actions in the preceding table are replaced with complementary actions after execution; these are Show-Cmds by Conceal-Cmds, Write-Cmds by Stop-Write-Cmds, Virtual by Absolute.

General philosophy on mixing keyboard and mouse button control is that, when possible, a command involving some typing is carried out completely at the keyboard, whereas commands involving mouse buttons are carried out completely with the mouse.

For example, to start a microprogram at some address, you normally type an address; then you could bug "Go" in the command menu, but probably "address;G" is more convenient because you won't have to lift your hand from the keyboard; ";G" are the command characters equivalent to bugging "Go".

Many commands are executed in overlays. When these get executed, the register display will turn off (The code for overlays resides where the display bit buffers would otherwise be.). During loading or execution of command files, the display is turned off to make the machine run faster.

Long-running commands normally display an "Abort" menu item. When this is bugged or when control-C is typed, the action terminates.

7. Keyboard

"=", "+", "-", "#", and "!" are legal symbol constituents in microprograms but will cause trouble for Midas if they appear in address symbols. "=" is an action character that will prettyprint the memory name and offset and the nearest address symbol less-than-or-equal to the value of the string on the input text line. "+" and "-" have their usual sum and difference meanings in evaluating input expressions. "#" (octal), "!" (decimal), and "%" (hexadecimal) may be inserted anywhere in a number to overrule the default radix; e.g., "#123" or "123#" will force the evaluation of the number "123" to be in octal. The default input/output radix for everything on D0 Midas is 8 (octal).

Lower case typein is converted to upper case by Midas, so avoid lower case characters in microprogram address symbols. You should write microprograms with the shift-lock key depressed or assemble them with the convert-to-upper-case assembly switch.

Typing ahead is legal until the character you type would cause execution of an action. After that, Midas will flush input and blink at you until the current action finishes.

At the end of an action, input text typed for that action is displayed on the input text line. This text remains valid and can be used as the arg for another mouse action. However, if you type any character (except control-A or backspace), the old input will be flushed before inserting the new character.

Keyboard editing characters are as follows:

control-A	delete last character
backspace	delete last character
control-Q	clear text line
del	clear text line

Other special keyboard characters are as follows:

control-C	abort the current action--equivalent to bugging the "Abort" command (only defined for actions that display "Abort")
control-Z	abort a command file
escape	repeat previous action (special for "Test" and "TestAll")
return	special following "Test" or "TestAll"
control-D	turns on the display (used during command files)
control-O	turns off the display (used during command files)
shift-swat	exit cleanly from Midas

The interrupt characters above are ineffective during loading, dumping, or comparing, which typically take between 2 and 20 seconds. Indefinite duration commands, such as "Go", "Test", etc. always monitor the keyboard, so control-C can be used to terminate them.

Control-Z, control-D, and control-O are intended for use during command files. However, these characters do not take effect until the command file executes a command such as "Go" which monitors the keyboard. There is no way to abort a command file and give control back to Midas safely except during a "Go" or other long-running command. This is not expected to be a problem because commands are executed quickly.

After interrupting a "Go" with control-C or control-Z, proceeding with ";P" or ";G" will succeed except when you have smashed the machine state by writing a register or in some other way or have interrupted an instruction from which continuation is impossible.

Although command menu items "SS," "Go," "Continue," "Break," and "UnBreak" are provided, the keyboard characters equivalent to these are usually more convenient.

8. Command Files

Command files (default extension ".Midas") are normally executed either by typing "Midas filename" to the Executive or by bugging a file name in the subsidiary menus put up by "Run-Prog" or "Read-Cmds". Alternatively, you may type a file name first, then bug one of these actions (If you type a file name after the subsidiary menu is put up and then bug "Abort", the command file will also be executed; it is not clear whether this is a bug or a feature.).

File names in these sub-menus are contained in the files Midas.Programs and Midas.UserPrograms, each of which has a list of file names separated by blanks, commas, or carriage-returns. Midas.Programs is part of the standard Midas release; Midas.UserPrograms is an optional file that a user of Midas can prepare with his own stuff. The names must be UPPER-CASE. These lists serve two purposes: building file FP's to speed OpenFile, and preparing the menu items for "Run-Prog" and "Read-Cmds".

If the file name contains no extension, then hint FP's will be built for both name.MB and name.MIDAS and name will be put in the "Run-Prog" menu. (However, the hint FP's are not built unless the file exists, and the file name will not be put in the "Run-Prog" menu unless name.MIDAS exists).

If the name ends in "*", a quick OpenFile table entry is made for name.midas and the name will appear in the "Read-Cmds" menu.

If the file name contains an extension, then it will be put in the quick OpenFile table, but won't appear in the "Run-Prog" or "Read-Cmds" menus.

Since Midas builds a table of file FP's during its initialization, when you edit a command file or modify a .MB file, you should reinitialize Midas by typing "Midas/I". When you add new command files or .MB files you should update the "Midas.UserPrograms" file appropriately and do "Midas/I".

"Read-Cmds" executes the actions stored in the command file; it is frequently used to modify the display in various ways by executing command files that show collections of items that are of interest.

"Run-Prog" first clears the symbol table and restores the display to its initial arrangement; then it executes the actions in the selected command file; "Run-Prog" is used to completely change contexts--to run a new microprogram, for example.

Generally, there is one "Run-Prog" command file for each hardware diagnostic, with the same name as the diagnostic, e.g.:

dgbasic.mb	the diagnostic;
dgbasic.midas	the command file.

A command file following this convention loads the diagnostic into the microprocessor and displays various registers of interest when the microprogram is in use.

The command-file facility is actually an (awkward and limited) programming language. The collection of actions discussed below is being developed so that command files can monitor diagnostic microprograms, collect and report error information on an output file, or direct the sequence of diagnostic microprograms according to hardware failures that are observed.

For system microcode, command-files can be used to control auto-restart and failure diagnosis.

Command files can be nested with "Run-Prog" and "Read-Cmds" subject to the following RESTRICTIONS:

(1) [Maxc2 only] "AltIO" terminates command files (i.e., upon return to Midas from AltIO the command file will not be continued).

(2) Nesting is limited to 8 levels (a parameter that could be increased if more levels are needed).

A number of actions, some of which cannot be given interactively, are useful in command files. These, not given in the table earlier, are shown below. The first table is for actions that operate on name-value menus (A0 ... C19); the second table for command menu (X) actions.

Table 2: Command File Name-Value Actions

Text Arg	Action	Comments
Address	Addr	Button actions as discussed earlier.
Value	Val	Button actions as discussed earlier.
	A+1	Increment memory address, as discussed earlier.
	A-1	Decrement memory address, as discussed earlier.
NCols	FillC	Fill name-value menus beneath the one selected with consecutive addresses starting at the address contained in the selected menu.
	Oct	Display address offset and value in octal.
	Dec	Display address offset and value in decimal.
	Hex	Display address offset and value in hexadecimal.
	Num	Display value numerically.
	Sym	Display value symbolically.
	Search	Display value as an address symbol plus offset in the appropriate memory.
Value	SkipE	Skip the next command if the input text evaluates equal to the contents of the register or memory word displayed. The input text is evaluated exactly as though it were to be stored into the register displayed in that name-value menu, so if the value displayed has several fields, the input text must also have several fields.
Value	SkipG	Skip if input text greater than the contents of the item in the name-value menu (unsigned compare).
Value	SkipL	Skip if input text less than name-value item.
Value	SkipNE	Skip if input text unequal to name-value item.
Value	SkipLE	Skip if input text less than or equal name-value item.
Value	SkipGE	Skip if input text greater than or equal to name-value item.

Table 3: Command File Command Actions

Text Arg	Action	Comments
Octal no.	Skip	Skip N following commands, where N is the value of the input text.
.Tag	Skip	Skip following commands until one is encountered with the label ".Tag".
Octal no.	BackSkip	
.Tag	BackSkip	Reset to byte 1 of the command file, then skip.
Octal no.	Return	Return out of current command file, then skip (".Tag" form is illegal for Return.).
	DisplayOn	Turn on the display, so that effects of subsequent commands can be observed. The display is initially off for a command file.
	DisplayOff	Turns off the display.
Octal no.	TimeOut	Input text is evaluated to a 32-bit octal number of msec at which to abort the immediately following command, if it has not finished by then. This is intended for use before "Go" and other commands which might hang indefinitely. If the timeout occurs, Midas will skip the command after the "Go". TimeOut also turns on the display, necessary because the machinery which checks for timeout is only active with the display on. Note that TimeOut is *required* before the actions *'ed in the table on page 4 and is *illegal* before other commands; Midas will complain if you do not use TimeOut appropriately.
	Confirm	Supplies confirmation for the command which follows (which should be one of the commands requiring confirmation).
File name	OpenOutput	Opens an output file (default extension ".Report") on which text can be written.
File name	AppendOutput	Append to an output file (default extension ".Report")
	CloseOutput	Closes the output file.
[text]	WriteMessage	Writes the contents of the input text buffer on the output file. Note that if any text follows the WriteMessage, that text up to but not including the <cr> is what gets written. However, if <cr> immediately follows WriteMessage, then the contents of the input text buffer left by the previous command get written. "~" is translated into <cr> and "\" into a blank.
	WriteDT	Appends the current date and time to the output file.
text	ShowError	Displays the text arg on the command line, turns on the display if it was off, and queries with "Abort" and "Continue" menu items.
--	DumpDisplay	Writes the current display image on the output file.
text	PrettyPrint	Evaluates text to a memory address, register name, or memory name; writes this name on the output file; then pretty-prints the value on the output file exactly as it would be pretty-printed on the comment lines if the item were displayed in one of the name-value menus and middle-buttoned.
File name	WriteState	Used by Midas initialization to create the Midas.Dtach and Midas.RunProg files--users shouldn't use this action.

9. Syntax of Command-file Actions

The syntax of a command-file action is as follows:

[["."<tag>]$" "]<buttons>$" "<menu>$" "<action>[$" "<text>]["";"<comment>]<cr>

where the "[]" denote that the ".tag", input text, and ";comment" are optional. $" " denotes a sequence of one or more blanks and tabs.

If the first character on the line is a ".", then the characters after that are a label or tag which may be used as the argument for the "Skip" or "BackSkip" actions given in the table earlier.

<buttons> may be any combination of the letters "L" (left-button), "M" (middle-button), and "R" (right-button); these are the buttons released to execute the action. These may appear in any order.

<menu> is the menu name in which the action is executed ("X" for the command menu, "A0"..."A19", "B0"..."B19", and "C0"..."C19" for name-value menus).

<action> is the text name for one of the actions (upper/lower case must match the definition).

<text> is the text typed on the command line, which may be anything except a ";".

> Note that if a *single blank* terminates <action> and if no input text argument is given, then input text left-over from the preceding action will be used. This allows text from a right-button action over a value to be used in a following action (e.g., in WriteMessage or to store the value into another reigster). However, one or more extra blanks will reset the input text, so the action is executed with null input text.

";" begins a comment, which may be omitted.

<cr> (carriage-return) terminates the action.

To find out what text should be put in command files, you can bug "ShowCmds" in the command menu. This will cause the command file text for each command to be displayed on the command comment line as the mouse selects it (You don't have to execute the command to see the equivalent text.). This text is complete except that the mouse button which executes the command isn't shown unless you depress the mouse button. To terminate "ShowCmds", bug "StopShow" (which appears only when "ShowCmds" is in progress.).

You can prepare a command file (default extension ".Midas") by typing a file name and bugging "Write-Cmds". This causes text for subsequent commands to be put on the file. When you are done with this, bug "Stop-Write-Cmds" to close the file. ("Stop-Write-Cmds" is in the command menu only when a command file is being written.). Exiting from Midas also closes the output file.

You will probably want to edit out your goofs with Bravo after the command file is written.

In addition, you will have to insert "Confirm" before actions which require confirmation and modify the "TimeOut" stuff which Midas uses to surround actions which might hang indefinitely (See the table given earlier for the actions that require this.).

Here is a sample command file:

```
L X Load DG1;                  Equivalent to typing "DG1" and bugging "Load" in the command menu
L A0 Addr CTASK;               Examine the "CTASK" register in name-value menu A0
L A0 Val 0;                    Change the value in CTASK to 0
L A1 Addr RTEMP;               Examine the address "RTEMP" in menu A1
L A1 SkipE FOO+3;              Skip the next command if RTEMP contains the value FOO+3
L X ShowError RTEMP not loaded correctly
L A2 TPC 0;                    Examine the TPC register for task 0 in menu A2
L X TimeOut 2000;              Abort the following command if it hasn't finished in 1.024 sec.
L X Go START;                  Begin microprogram execution at address "START"
L X Skip 1;                    Skip the next command if "Go" halts before timeout
L X ShowError START;G failed;  Show an error message
```

10. Registers and Memories Known to Midas

Table 4: Memories

Memory	*Width (octal)*	*Length (octal)*	*Notes*	*Comments*
IM	100	10000		Control store (virtual).
IMX	44	10000	4,5	Control store (absolute).
RM	20	400	4,5	
T	20	20	1	Primary task-specific temporary register.
TPC	20	20	1,2	Task-specific subroutine return link.
VM	20	2^{22}	4	Main storage (addressed through the MAP)
MAP	20	2^{14}	4	Maps VM to absolute storage.
BP	100	402	3	Breakpoint information used by Midas.
MIM	60	1000	3	Holds microinstructions used by Midas.
MDATA	und.	10	3	BITS-CHECKED etc. for testing.
MADDR	40	20	3	LOOP-COUNT etc. for testing.

1. Task-specific register
2. Virtual/absolute stuff applies
3. Fake memory--artifact of stuff in Midas
4. Appears in Test menu.
5. Appears in TestAll menu.

Table 5: Registers

Register	*Width (octal)*	*Notes*	*Comments*
APCTASK	4		Next task.
APC	14	2	Current task's subroutine return link or next task's PC.
CTASK	4	3	Task for which "T 20" and "TPC 20" apply.
CIA	14	2	Current instruction address.
CYCLECONTROL	10		See HW manual.
PAGE	4		Current IMX page.
PARITY	4	1	See HW manual.
BOOTREASON	10	1	See HW manual.
PCXREG	4		See HW manual.
PCFREG	4		See HW manual.
DBREG	6		See HW manual.
SBREG	6		See HW manual.
MNBR	20		See HW manual.
ALURESULT	4		See HW manual.
SALUF	10		See HW manual.
SSTKP	10		See HW manual.
STKP	10		See HW manual.
MEMSYNDROME	20		See HW manual.
CALLER	14	2	Shows contents of APC-1, (address of last subroutine call); cannot infer this from APC in virtual mode.
AATOVA	20	3	Translate absolute address to virtual

1. Read-only to Midas.
2. Virtual/absolute stuff applies
3. Fake register--artifact of stuff in Midas

Most registers and memories listed above correspond to ones discussed in the "D0 Hardware Manual". Others are discussed in the sections which follow.

MDATA and MADDR memories contain words used to report or control the activity of the

"Test" and "Test-All" actions discussed later. MADDR also contains COMM-ER0, COMM-ER1, COMM-ER2, and BOOT-ERR (error-reporting), which will be discussed later.

The BP and MIM memories are not expected to be of interest to programmers. BP holds the information about breakpoints that is manipulated by the breakpoint actions discussed later. MIM holds microcode overlays that are written into the D0 microstore to operate the hardware.

For approximately all registers and memories that contain 16-bit quantities, Midas will evaluate input of the form "m,,n", storing the value of "m" into bits 0:7 of the word and the value of "n" into bits 8:15.

On D0, the items that accept "m,,n" are RM, T, VM, MAP, and MNBR.

11. The IM Memory and Virtual Addresses

Because the placement transformations performed by MicroD make it difficult to correlate microstore locations with positions in microprogram source files, the Dorado and D0 Midas implementations use a map to transform virtual addresses produced by Micro into absolute microstore locations produced by MicroD.

Two memories, IMX and IM, each show the microstore. IMX is absolutely addressed; IM virtually addressed. When you fire up Midas, IM is "empty"; when you load a microprogram, IM is filled with consecutive instructions from your source file, irrespective of where MicroD decides to place these; the "value" displayed for an IM word includes both the absolute address assigned to it and the microinstruction.

In other words, if your microprogram is 10 words long, the meaningful part of IM is only 10 words long. In this case, if you examine IM addresses greater than 7, the printout will show an absolute address of 7777 and zeroes for the rest of the value.

Midas will not allow you to modify the mapping between virtual and absolute addresses interactively--you can only do this by loading a microprogram.

To facilitate dealing with virtual/absolute correspondences, Midas has a mode switch that controls handling of registers and memories that normally contain microstore addresses. When you fire up Midas, the display is in absolute mode and the "Absolute" action appears in the command menu; when you load a microprogram, the display switches to virtual mode and the "Virtual" action appears in the command menu. Test actions will switch to absolute mode. The *current* mode always appears in the command menu.

In virtual mode, the display shows the virtual equivalent for the value in any register that normally contains a microstore address. When the value is outside the virtual memory, it prints as 7777. To find the absolute value in this case, you have to switch to absolute mode.

On D0 the registers affected by this are TPC, APC, CIA, and CALLER.

The general idea is that, if you suspect a hardware problem in the control section, you might work in absolute mode, but in all other situations when a program is loaded you will work in virtual mode, and the complications created by scrambled instruction placement will be concealed.

A fake register called AATOVA converts absolute addresses to virtual. For example, copying the value in some RM word into AATOVA will show the virtual equivalent; this is useful when

return links are saved in RM words.

The convenient way to use AATOVA is to first right-button the value from an RM word that contains a return link (which puts the value on the input text line); then left-button the value into AATOVA, which will prettyprint the virtual address on the comment lines.

12. Registers and Memories that Contain Microinstructions

The IMX, IM, and MIM memories all contain microinstructions. A middle-button action over the value will print these symbolically on the comment lines.

The value for an IM word is shown as four fields on the display:

14_8-bit absolute address;
bits $40\text{-}43_8$ of microinstruction;
bits $0\text{-}17_8$ of microinstruction;
bits $20_8\text{-}37_8$ of microinstruction.

The format of the bits is as shown in the hardware manual. You will note that the RSEL and JA fields are scrambled in this arrangement; each of these has two bits in the main part of the microinstruction and two other bits in the 4-bit extension, and two of the RSEL bits are inverted. This, together with the numerous fields in each microinstruction, makes octal interpretation and modification of microinstructions somewhat tedious, so the symbolic pretty-print and input forms discussed below should generally be used.

IMX and MIM are like IM, but the 14_8-bit absolute address field is absent from IMX.

The MIM memory is an array in Alto core that contains microinstructions used by Midas when operating the hardware; it should ordinarily be of no interest to users.

Note that the microinstruction pretty-print procedure does not have available all of the information that the microassembler had when you assembled your program, so the printout is not always beautiful. The following are deficiencies you should be aware of:

From the hardware manual, you will remember that the interpretation of some instruction fields depends upon the task executing the instruction, so Midas will disassemble correctly only when it is able to deduce the task that executes the microinstruction.

There are many possible assembler macros that you might use to generate constants to control the shifter; for an instruction that does this, Midas might not choose the form you used in the source file.

When you want to modify a microinstruction, a special form of input is available as follows: The first character on the input text line should be "(" to change the values of several fields in the instruction without clobbering other fields, or "[" to reconstruct the value beginning with a no-op microinstruction. This is followed by a number of clauses of the form "Field←integer" separated by blanks and/or commas. The legal field names are:

RSEL, JA, MEMINST, F2, and JC for all instructions;

RMOD, ALUF, BSEL, F1, LR, and LT for regular instructions only; and DF2, TYPE, and SRCDEST for memory instructions only.

In addition to "field←value" clauses, Midas interprets the standalone clause RETURN, and branch clauses of the form GOTO[addr], CALL[addr], GOTO[addr,bc], GOTOP[addr], or CALLP[addr]. In these "addr" is interpreted as a virtual IM address in virtual mode or as an absolute IMX address in absolute mode; "bc" is a symbolic branch condition.

The "addr" argument to GOTO, CALL, GOTOP, and CALLP will usually be a simple integer in absolute mode but may be an expression such as FOO+3, where FOO is an IM address, in virtual mode. Midas will give an error if the target for GOTO or CALL is off-page; in the event that an off-page branch is legitimate because the predecessor did a LOADPAGE, then this error check will thwart you--you have to use GOTOP or CALLP, which do not check for off-page, in this situation.

On a conditional GOTO, Midas will check that the target is at an odd location for a "regular" branch condition, or at an even location for an "inverted" branch condition. The branch conditions are as follows:

Regular: ALU#0, CARRY, ALU<0, NOH2BIT8, R<0, RODD, NOATTN, MB, INTPENDING, NOOVF, BPCCHK, SPARE, QW0, and TIMEOUT;

Inverted: ALU=0, NOCARRY, ALU>+0, H2BIT8, R>=0, REVEN, IOATTN, NOMB, NOINTPENDING, OVF, NOBPCCHK, NOSPARE, NOQW0, and NOTIMEOUT.

For parts of the microinstruction other than the control clause, Midas requires you to use the awkward "field←value" form.

13. Task-Specific Registers

Midas treats all task-specific registers (T and TPC) as 20-word memories. In other words, "T 6" is the T-register for task 6.

In addition, a special kludge allows you to display the 21st word (i.e., "T 20" or "TPC 20") and have that be interpreted as the register for the *currently selected task.* The currently selected task is the value in CTASK; CTASK is initialized to the task which broke at breakpoints.

In other words, when a microprogram halts at a breakpoint or because of a mouse-abort, CTASK becomes the word number for the "T 20" and "TPC 20" items on the display; if CTASK contains 6, these will show values for task 6. You can see the registers for another taks by modifying CTASK on your display. CTASK is also the task started by "Go", "SS", etc. as discussed later.

APC contains either the subroutine return link for the current task or the PC for a task about to be reactivated; when it is a subroutine return link, it holds the location of the last CALL or'ed with 1. In virtual mode, because of the scrambled instruction placement, this will not readily translate into the location of the caller, so Midas provides a variant of APC named CALLER, which holds the value in APC less 1; in virtual mode this will show the IM address of the last CALL.

14. Memory System Registers and Memories

VM accesses the virtual memory using the current contents of the MAP. Midas does not provide any direct method of accessing storage; the user has to setup MAP with appropriate values and then use VM to do this.

MAP, MEMSYNDROME, etc. *to be filled in*

15. Loading Programs

The "Load", "LdSyms", and "LdData" actions are used to load micro-binary files into the machine. These actions are executed by first typing a list of file names (default extension ".mb") separated by commas, then bugging "Load" or "LdSyms" (typing ";L" is equivalent to bugging "Load"). These actions require confirmation by <cr>, "Y", or "." iff a previously-loaded program is being overwritten; in a command file where it is not known whether or not another program is being overwritten, a "Confirm" action should precede the load action, as discussed earlier.

"Load" loads the entire .mb file--symbols into the Midas symbol table, data into the hardware, and breakpoints into the BP memory.

"LdSyms" loads only the address symbols and IM mapping table from the .mb file; the BP memory is not loaded and data are not loaded into the hardware.

"LdData", (in command files but not available interactively), loads data blocks and the BP memory from the .mb file; symbols and the IM mapping table are not loaded.

> On D0, the MADDR and MDATA memories are treated as exceptions by "LdData"--symbols for these are loaded anyway.

Midas uses several 1024-word core buffers (about 12 on D0 Midas) and the Swatee file to manage its symbol table and virtual memory mapping information; the largest existing programs use 10 buffers for VM information and about 20 more (out of 64 available on Swatee) for symbols. For nearly all symbol and VM accesses, Midas will reference only one or two symbol blocks, so there should be no appreciable slow down when handling large programs.

> The symbol table management algorithm used by Midas is an extremely fast merge that works well when the symbol table is nearly empty at the onset of a load but suffers somewhat from block fragmentation when the initial symbol table has many items.
>
> To avoid fragmentation, don't load one microprogram on top of another--use "Run-Prog" to reset the symbol table, then do the "Load". It is also a good idea to assemble microprograms as a single .MB file. Although Midas can load multiple .MB files (typed as a list separated by commas), this will fragment the symbol table and cause extra thrashing.
>
> These recommendations follow because Midas takes advantage of alphabetical address ordering in .MB files to pack its symbol buffers nearly full. But when subsequent files are loaded, the symbol buffers will fragment to about half-full, symbol buffer swapping will result, and symbol searches will be longer.
>
> Midas uses the symbol table in two ways: looking up the value of a symbol, requiring at most one disk access; and searching for the symbol in a particular memory which best matches a value, requiring at most one access for RM or at most two accesses for IM address symbols; the best matching value for addresses in all other memories is determined by scanning every block. Searching every block requires about (.22 seconds * no. symbol blocks) - (.15 seconds * no. blocks in core) or about 2.9 seconds for the largest program thus far. However, since best matches for the two most important memories are obtained quickly,

it will rarely be necessary to wait for a search.

In most situations where a "Load" is going to be done, many other actions will also be carried out to setup the display appropriately for the program. For this reason, you will ordinarily want to define a command file that does all these other actions as well as the "Load" and you will ordinarily do "Run-Prog" on this command file; direct use of "Load" in the command menu will be rare.

Midas/MicroD do not handle microprograms with overlays conveniently. At present, the system microcode consists of an initial microstore image that contains both some resident code and initialization code; the initialization code is executed and then overwritten by the rest of the resident system. Midas/MicroD do not provide any clever features for setting up the symbol table and IM mapping table correctly in this situation. One method of handling this situation is to create a "Run-Prog" command file which does the following:

1) A "Load" on the resident+initialization; then a "Go" at the starting address, which runs up to a breakpoint at the end of initialization.

2) A "Run-Prog" on another command file; this clears the breakpoint, IM mapping, and symbol tables. The command file does a "Load" on the original resident+rest of resident that replaces the initialization code and returns to the outer command file.

3) A "Go" at the starting address (or a "Continue" from the initialization breakpoint) to start the system which then runs until it fails or halts.

Assuming that you are somehow able to build the two .Mb files needed by this sequence (It is unclear how you will do this.), you will wind up with Midas containing the correct symbols and IM mapping table for debugging.

16. Dump and Compare

Both "Dump" and "Compare" require confimation by <cr>, Y, or "." They accept the name of a microprogram (default extension ".mb") on the input text line. If the input text line is empty, then the file name is defaulted to the name of the program last loaded.

"Dump" deletes forward reference fixups left by Micro (which never occur on Dorado or D0 because MicroD does these) and compacts both data and addresses to use less disk space and load more quickly later. Dumped files are about 20% smaller and can be loaded 10% to 15% faster than undumped files, so it is desirable to load and then dump .mb files that will be used widely.

Also, if undumped .MB files contain forward references, they cannot be used with "Compare" (no problem on Dorado or D0).

Note that *only memory words loaded by Load are dumped*--you cannot patch unused locations, dump the program, and expect the patches to survive. (You might assemble extra locations as a patch area with your microprogram, so that you can patch and dump during debugging, but placement constraints will be difficult to satisfy.)

"Compare" compares data currently in storage against data in the file and reports differences on the Midas.Compare file.

In microprograms, avoid loading initial values into memory words modified during execution. The usefulness of "Compare" is enhanced when programs are clean, because no fictitious errors will be reported.

For diagnostics, "Compare" can report what has been smashed when something goes off the deep end--this has frequently been helpful.

Following system microcode crashes, "Compare" may provide the only clue about the nature of an intermittent storage failure.

17. Break, UnBreak, ClrAddedBPs, ClrAllBPs, and ShowBPs

"Break" inserts a breakpoint in the IM or IMX address typed on the input text line. The address must be typed--there is no default break address. You will normally find it faster to type "address;B" to insert a breakpoint.

"UnBreak" removes a breakpoint. If no text is typed, the address defaults to the breakpoint that caused the last program halt or to the address of the last breakpoint inserted. You will normally find it faster to type "address;K" or ";K" to remove a breakpoint.

"ClrAddedBPs" removes all the breakpoints inserted since the last "Load" and prettyprints the addresses of the first 10 removed. "ClrAllBPs" clears all breakpoints, including those that were loaded witht he program. "ShowBPs" prettyprints the addresses of all breakpoints added since the last "Load."

Breakpoints are implemented by replacing the broken instruction by a special breakpoint instruction. When the D0 is halted, IMX contains the unbroken instructions, and Midas remembers which places contain breakpoints; when you continue your program with "SS," "Go," or "Continue," Midas saves the instructions in its table (the BP memory), and stores breakpoint instructions at those places; when the program halts, Midas restores the contents of IMX.

Single-stepping is also implemented with breakpoints; Midas determines one or both possible successors to the instruction being single-stepped, plants breakpoints there, starts the machine, and then undoes the breakpoints after the machine halts; BP's 0 and 1 are used for this purpose.

A breakpoint can be put on any instruction. However, there is a limit of 254 user breakpoints; also, there are some restrictions on continuing discussed in a later section. You may be unable to continue from breakpoints on some instructions.

18. Go, SS, and Continue

These are actions that result in the microprocessor resuming or starting execution at the selected address. "Go" and "SS" accept an optional address argument on the input line that must evaluate to an IM or IMX address; a simple number is defaulted to an IMX address in absolute mode or an IM address in virtual mode. If the optional argument is omitted, Midas will continue from the last break. "Continue" always continues from the last break, ignoring any text on the input text line.

The keyboard equivalents for these commands are ";G" for "Go"; ";S" or ":" for "SS"; and ";P" or ";C" for "Continue."

When you start at a new address, the value in CTASK (lower left-hand corner of the normal

display) is the task activated. You must change CTASK on the display before initiating execution for a different task.

When the microprocessor halts after a breakpoint, due to an error, or because you aborted, Midas prints the location of and reason for the halt and saves the information that it needs to continue. The form of the printout is "task:address". Subsequently, if you attempt to continue, Midas restores the hardware as nearly as possible to its state at the break before continuing.

There are some complications surrounding Midas' ability to restore the state of the program, after doing other things, so that continuation is possible. These are discussed in the next section.

19. When Registers are Read/Written--Restrictions on Continuing

When a microprogram halts at a breakpoint or due to a mouse-halt, Midas has two objectives: to read the contents of registers and memory addresses so that they may be shown to the user, and to be able to continue from the interrupt or breakpoint. In terms of how the Midas read/write procedures work, there are three cases:

Registers: The Kernel program running on D0 saves registers in RM words reserved for the purpose; Midas reads these special RM locations into a block of Alto storage when the machine halts or when you do a "Boot" action. Subsequently, Midas displays the values from Alto storage, and, when a register is written, modifies the Alto storage. The D0 hardware is not affected by any change in register values until you resume or start your D0 program. At that time, the block of Alto storage is rewritten into the Kernel's special RM locations, and the Kernel will transfer these values into the registers just before releasing control to your program.

Memories: IM/IMX, RM, T, TPC, MAP, and VM addresses are read and written directly; whenever you modify a word in one of these memories, Midas will write it (through the Kernel); Midas always reads the values from the hardware, never from remembered values in Alto core.

Artificial registers and memories: For CTASK, AATOVA, BP, MIM, MDATA, and MADDR, Midas modifies/reads the Alto storage containing the value, so the D0 hardware is not affected.

However, whenever any register, memory word, or artificial register or memory word is modified, Midas rereads the value for every item on the display, going left-to-right and top-to-bottom through the display. This is unimportant as long as the D0 hardware is functioning correctly, but if the hardware is unreliable, then displayed values of memory words may change, so be wary.

There are a number of situations that may prevent continuation from a breakpoint or interrupt; Midas warns you about some of these when you try to continue but does not warn you about others. Some of the ones that Midas does not warn you about are as follows:

> Input/output tasks were not serviced properly due to the delay at the breakpoint, so these are not continued correctly;
>
> You break on any of the three instructions involved in the "bypass kludge," when the instruction after a memory operation expects to read the result of the memory addition instead of the value for which write is pending into T or RM.

Some situations that Midas does warn you about are as follows:

> You broke at the instruction after a LoadPage. This happens either because you break on the instruction

after a LoadPage or because you break on the LoadPage instruction itself and Midas breaks on the instruction after the LoadPage when restarting.

20. Hardware Failure Reporting

Midas checks for several kinds of hardware errors and reports them in COMM-ER0, COMM-ER1, and COMM-ER2, which are addresses in the MADDR memory; these are shown in the upper right-hand name-value menus of the normal Midas display. Values have two 16-bit fields; each field counts errors of some type and can be prettyprinted for interpretation. Midas does not print any special messages after these errors--the user will have to notice when they change.

A "Boot" action is carried out by first loading selected IMX words from a ROM; Midas can cause the D0 hardware to do this through its Diablo Printer interface, as discussed in the hardware manual. When this part of the boot finishes, Midas transmits the Kernel into IMX by communicating with the boot loader. If Kernel transmission is successful, Midas then starts the Kernel.

Four possible communication errors may be detected during Kernel transmission. If one of these failures occurs, Midas reports the failure in BOOT-ERR (an address in the MADDR memory) and reattempts the boot, not giving up until the boot has failed 10 times. BOOT-ERR is shown on the display as two 16-bit fields; the left-most field shows how many words were transmitted before the (last) failure occurred; the right-most field contains four four-bit nibbles that count the number of failure occurrences for each of the four reasons.

As soon as the Kernel has been successfully transmitted, Midas will attempt to start it running; if this fails Midas will immediately report a failure without retrying.

MEMSYNDROME and BOOTREASON registers report failures detectd by the D0 hardware, as discussed in the hardware manual.

21. Testing Directly From Midas

"Test" and "TestAll" allow the target machine to be tested directly from Midas. Although diagnostic firmware can test faster and more thoroughly than is practical from Midas, Midas direct testing permits the hardware to be checked out well enough to get basic diagnostics loaded and started. On Maxc1, which had no direct testing in Midas, many hardware failures of the "nothing works" variety were harder to fix than on Maxc2 and Dorado, where Midas test software is available.

However, on D0 and M68 implementations of Midas, the test features in Midas are of doubtful usefulness because the hardware is accessed through communication with a small "Kernel" microprogram that only works when most of the hardware is functional.

On D0, only IMX and RM are presently testable, but the address ranges are limited so as not to overwrite the parts of these memories used by the Kernel. Neither of these actions is expected to be useful because most failures in these memories will prevent the Kernel from running. They are described here anyway.

Data patterns for test actions are determined from the first subsidiary menu, as follows:

Table 6: Test Data Pattern Actions

ZEROES	All-zeroes data
ONES	All-ones data
SHOULD-BE	Constant test pattern equal to value in SHOULD-BE
CYC1	Vector of the same size as the register containing zeroes with a single one-bit cycled left one position each iteration
CYC0	Cycled zero in vector of ones
RANDOM	Random numbers
SEQUENTIAL	0, 1, ..., sequential numbers
ALTZO	Alternating all-ones and all-zeroes patterns
ALT-SHOULD-BE	Alternating contents of SHOULD-BE with its ones-complement

The CYC0, CYC1, and SEQUENTIAL patterns vary according to the size and arrangement of the data vector for the item being tested. CYC0, for example, starts off with leading 1's and a 0 in the right-most bit of the data vector. The 0 is shifted left (bringing in 1's to its right) each iteration; when the 0 is shifted out of the left-most bit in the data vector, the vector is reinitialized to leading 1's and a 0 in the right-most bit. The CYC1 pattern is like CYC0 with 1's and 0's interchanged. The SEQUENTIAL pattern is initialized to 0 and is incremented by 1 in the right-most bit of the data vector each iteration.

This treatment of CYC0, CYC1, and SEQUENTIAL patterns is conceptually correct for items that are described inside Midas by dense, left-justified data vectors whose bits are displayed left-to-right on the screen. On D0 all testable items are handled this way.

Testing is controlled/described by 12 addresses on the display as follows:

Table 7: Test Items in the Name-Value Display

SHOULD-BE	On a failure, the correct data; after control-C or Abort, the next pattern.
DATA-WAS	On a failure, what the data was; after control-C or Abort, the data read last time.
BITS-CHECKED	Mask of bits checked (see below).
BITS-PICKED	Union of bits that should have been 0 but were erroneously 1 during testing. This accumulates failure information when you continue a Test using <escape> or <cr>.
BITS-DROPPED	Union of bits that should have been 1 but were erroneously 0.
LOOP-COUNT	32-bit iteration count at which failure occurred or after which the test was aborted.
NFAILURES	32-bit count of test failures.
Memory tests only	
LOW-ADDR HIGH-ADDR CURRENT-ADDR ADDR-INC	32-bit addresses: If ADDR-INC (normally 1) is positive, the test starts at LOW-ADDR and advances through the memory in steps of ADDR-INC until CURRENT-ADDR is greater than HIGH-ADDR. If ADDR-INC is negative, the test starts at HIGH-ADDR and goes by steps of ADDR-INC until CURRENT-ADDR is below LOW-ADDR. CURRENT-ADDR contains the last address tested.
ADDR-INTERS	Intersection of address bits where failures were detected.
ADDR-UNION	Union of address bits where failures were detected.

SHOULD-BE, DATA-WAS, BITS-CHECKED, BITS-PICKED, and BITS-DROPPED are addresses in the MDATA memory; LOOP-COUNT, NFAILURES, LOW-ADDR, etc. are addresses in the MADDR memory. These two memories (which are tables in Alto storage) exist on all versions of Midas that implement the test actions.

The handling of the MDATA memory is complicated by the fact that items in this memory have to be shown in the same format as the memory or register being tested. This is accomplished as follows: When the selected test item is different from the last, the width and print-format of MDATA are set to be identical to the new item; in this case BITS-CHECKED is initialized to test all bits in the new item. Then when the test is aborted or halts due to a failure, the display of BITS-CHECKED, etc. is identical to that of the item tested. The user may then modify BITS-CHECKED and continue, restart, or free-run the test, as discussed below; in this case the item tested is identical to the last item tested, so BITS-CHECKED is not reset.

The handling of MADDR is also tricky. ADDR-INC is allowed to be any value except 0; if it is 0, Midas will reset it to 1 before testing. When HIGH-ADDR is initially greater than the largest legal address in the memory, it is reset to memlength-1 prior to testing. Then if LOW-ADDR is greater than HIGH-ADDR, it is reset to 0 before testing. When the selected memory differs from the last item tested, and when the length of the memory is less-than-or-equal to 10000_8 words long, Midas will reset LOW-ADDR to 0 and HIGH-ADDR to memlength-1 prior to testing. This is done because a common operational error is failure to reset the address range when switching from one memory test to another. However, Midas does not reset the address range for very long memories because they are normally tested with small address ranges that cannot be predicted in advance--full-length testing of long memories from the Alto is so slow as to be impractical.

"Test", after showing the data-pattern menu, shows a menu of register and memory names and other test names, and executes a test of the one you select until the test fails or you halt the test from the keyboard.

The testable registers and memories appear in the second sub-menu for the "Test" action. Provision is also made for other machine-dependent tests, but there aren't any implemented for the D0.

<esc> will continue a register or memory test that has halted; it restarts an OtherTest that has halted.

<cr> will continue a register or memory test that has halted but will free-run the test rather than halting on the next failure. While free-running, LOOP-COUNT and NFAILURES are reported continuously on the display, and BITS-DROPPED, BITS-PICKED, ADDR-INTERS, and ADDR-UNION accumulate failure information. When you stop the test by bugging "Abort" or typing control-C, the accumulated failure information is displayed in these registers.

"TestAll" automatically loads BITS-CHECKED with a full-sized comparison mask prior to testing each item; memories are tested with LOW-ADDR = 0, HIGH-ADDR = memory length-1, and ADDR-INC = 1. It tests each register 200 times and makes 4 passes through each memory and each OtherTest.

22. Command Files Used With "RdCmds"

At the time this was written, the following command files were in use:

Table 8: Command Files

midas-tests	restore "normal" Midas display with the hardware testing items in the right display column.
svcrash	write the Midas display followed by a pretty-print of most registers on the file Crash.Report.
tpc	show 20_8 TPC registers in middle column.
t	show 20_8 T registers in middle column.

```
 AATOVA     177777                                COMM-ER0                  0 0
 CYCLECONTROL    0                                COMM-ER1                  0 0
 PCXREG          1                                COMM-ER2                  0 0
 PCFREG          2                                BOOT-ERR                  0 0
 DBREG           0   TPC 0          1234
 SBREG           0   TPC 1          5677
*MNBR            0   TPC 2           123
 SSTKP           0   TPC 3          7102
 STKP          302   TPC 4          2274
 ALURESULT       7   TPC 5          3331
 SALUF          11   TPC 6          1255
 T 20       123123   TPC 7          2771
                     TPC 10         1336
 TPC 20       7002   TPC 11         7774
*CALLER       7103   TPC 12         1177
*PAGE           16   TPC 13          444
 APC           7105   TPC 14         4055
 APCTASK          3   TPC 15          117   BOOTREASON                   1
 CIA           7104   TPC 16         6660   PARITY                       0
 CTASK            3   TPC 17         7403   MEMSYNDROME               0 0

Loaded: KERNEL

Go at 0:BEGIN, BrkP after 0:QERR+1 at 0:QERR+2


Exit Boot Run-Prog Read-Cmds Break UnBreak ClrAddedBPs ClrAllBPs ShowBPs Go
SS Continue Load LdSyms Compare Test-All Test Dump Show-Cmds Write-Cmds
Absolute

BEGIN;
```

Sample Midas Display

XEROX

PARC
14 June 1983

To: Microcode Developers

From: Edward Fiala

Subject: How to Debug Dolphin Microcode With Midas

Filed On: [Indigo]<D0Docs>DebugWithMidas.Bravo, .Press

This is a short description of how to debug Pilot, Cedar, or AMesa microcode with Midas. It handles any combination of microswitch/Star keyboards and CSL/LF monitors. Documentation for Midas can be found on [Indigo]<D0Docs>D0Midas.Press. You should read both the Midas document and the part of "Dolphin Booting and Maintenance Panel Codes" ([Indigo]<D0Docs>MPCodes.press) which describes booting.

Midas is a debugging program that runs under the Alto OS on a different machine from the one you are debugging. You can run it on an Alto or on the Alto partition of a Dolphin. In either case, you need a special cable to connect the printer interface of the Alto or Dolphin running Midas to the printer interface of the Dolphin being debugged.

If you don't already have a Midas disk, you will need to build one:

1. Spin up a clean disk on your Alto. Boot the NetExec and invoke NewOS. Use the long installation dialog and erase the disk. If you going to run Midas on a Dolphin, you don't have to do this--just make sure there are 2000 free pages on your Alto partition.

2. Obtain the following files from the place where you get Alto subsystems (e.g., [Maxc]):

Micro.run	(microcode assembler)
MicroD.run	(microcode loader)
Ftp.run	
Bravo.cm	and execute this command file
RunMesa.run	
Empress.run	
Find.run	
Waterlily.run	

3. Load D0MidasRun.dm and retrieve the other files below from [Indigo] with Ftp:

<D0Source>D0Lang.Mc	
<D0>D0MidasRun.dm	
<D0>Midas.programs	
<D0>MemErrors.midas	
<D0>MakeLoaderFile.bcd	(microcode boot file builder)

Because of file name conflicts, you can only have one of the following dumps of microcode sources loaded at-a-time (except that Initial can coexist with Pilot microcode if Pilot microcode is loaded last):

<D0Source>CedarUCode.dm	for Cedar microcode
<D0Source>PilotUCode.dm	for Pilot microcode
<D0Source>AMesaSources.dm	for AMesa microcode
<D0Source>InitialSources.dm	for Initial microcode

4. Call Ed Fiala or send message to Fiala.Pa if you have problems with these procedures.

There are several important limitations to the ways in which Midas can be used to debug a Dolphin. The first is that you must BEGIN with Midas. If instead you attempt to attach to a machine which is in an interesting state, then Midas will boot the machine while activating its Kernel microcode, and all RM registers and the TPC's for the tasks will be reset (possibly storage will survive booting, however).

The second limitation is that the various microcode systems that you run must all reserve space for the Midas Kernel and have an appropriate linkage between the fault handler and the Midas Kernel. Initial has an IMReserve for the Midas Kernel; if you assemble the Pilot microcode with WithMidas = 1 in GlobalDefs.Mc, then Pilot also has an IMReserve for the Midas Kernel. SDD Pilot microcode at the present time is built with WithMidas = 1, but the Cedar and Tor variants are not, and the normally released Alto emulator overwrites the Midas Kernel, so you will need to obtain or create special debugging versions of these to debug from Midas.

The third limitation is that you cannot activate Pilot directly from Midas by loading and running .Mb files. Instead, you must install the Pilot germ and microcode on the Dolphin SA4000, and then load the microcode by running the Initial microcode from Midas. The microcode installed on the SA4000 must be the SAME as that on your Midas debugging disk. In addition, you must have a Physical Boot Volume set.

Using Midas to Boot Pilot

Boot the Midas disk and type "Midas/i Pilot". The Pilot.Midas command file will first load Initial.Mb, insert a breakpoint at RamLoaded, and start Initial at SAPilotStart. If all goes properly, Initial will read the Pilot microcode and all of its overlays from your SA4000 into storage, and the breakpoint at RamLoaded will be hit after LoadRAM has loaded Pilot1 into the microstore; then Pilot1 symbols are loaded by RunProg and the RamLoaded breakpoint is reinserted in octal (at IMX 316); this is necessary because all previous symbols and breakpoint information (which pertained to Initial) have been flushed by Run-Prog; since the space occupied by the LoadRAM module in Initial has been IMReserved in Pilot, the RamLoaded symbol is no longer available.

The Pilot.Midas command file pauses at this point with a message saying that Pilot1 is loaded, and you may abort the command file or continue it. If you continue the command file, it will pause again at BootEmulators in Initialize.Mc after comleting all device initialization but before loading the Pilot2 overlay. If you continue from there, it will pause a final time after loading the Pilot2 overlay; if you continue from there, your physical boot volume will be started.

Assuming that you abort the first time the Pilot.Midas command file pauses, then you have the following options:

a) Kill the IMX 316 breakpoint and proceed, letting Pilot start normally. After it is running, you can get control with Midas control-C (or by selecting and left-buttoning "Abort") and establish a suitable debugging context as discussed below.

b) Insert any breakpoints in Pilot1 that you wish and continue. This allows debugging initialization and other Pilot1 microcode.

c) If you want to debug the terminal microcode, then the driver for the LF keyboard system is included in Pilot1, so you can insert breakpoints directly; for a CSL keyboard system (with either a CSL or LF monitor), you must proceed from the RamLoaded breakpoint--the next time you get to that breakpoint, the microcode for that terminal, obtained from the first or second overlay in PilotD0.Eb or CedarD0.Eb, will have been loaded into the microstore. See below for establishing a debugging context.

To debug Pilot2 microcode, you should continue the command file until it indicates that Pilot2 has been loaded; then abort it.

Displaying Pilot Microcode Symbols From Midas

When Pilot has been loaded and started as described above, you can setup a debugging context by doing Run-Prog on PILOT1SYMBOLS, PILOT2SYMBOLS, CSLSYMBOLS, or CSLFSYMBOLS. Run-Prog clears all previous symbols and breakpoints and loads symbols from one of the four Pilot overlays--it is impossible to have symbols from more than one overlay concurrently active. PILOT1SYMBOLS and PILOT2SYMBOLS also setup the middle column of the Midas display with registers generally of interest when debugging Pilot emulator code; CSLSYMBOLS and CSLFSYMBOLS setup the middle column with registers generally interesting when debugging one of the display drivers.

At any time you can change the contents of the Midas display by doing Read-Cmds on a variety of files in the menu with suggestive names. Among these are the following:

BBREGS	Displays BitBlt registers in middle column.
RDCREGS	Displays RDC registers in middle column.
GCDEBUG	Displays CedarGC registers.
MEMERRORS	Displays memory error registers in right column.
TX*	Various TextBlt registers.

Using Midas to Debug AMesa microcode

1. Obtain a special debugging version of the Alto emulator that contains an IMReserve for the Midas Kernel--normal Alto microcode overwrites the Kernel and cannot be used. [Indigo]<D0>NewAMesa.Mb with CSL keyboards or [Indigo]<D0>LFAMesa.Mb with LF keyboards are suitable; put one of these on your Midas disk.

2. Boot the Midas disk and type "NewAMesa;L"; this loads the Alto microcode. Start the Alto emulator with "EGO;G" "KGO;G" or "KGOP2;G" for ether boot, partition 1 boot, or

partition 2 boot, respectively. The Alto Executive will appear shortly on the screen.

Eprom Microcode

Sources and other files for the Rev-L EPROM microcode are on [Indigo]<D0Source>Proms>Rev-L>*. After you rebuild the .mb file for the EPROM microcode from the sources, you can debug it from Midas as follows:

1. Select Run-Prog on the Midas menu.

2. When the Run-Prog menu appears, select EPROM.

When SA4000Boot is loaded and the next menu appears, you can select "CONTINUE" to continue the boot sequence with the Initial microcode, obtained from the Initial microcode area of the SA4000.

Loading Initial

To load the Initial microcode, use the Initial command file:

1. Select Run-Prog on the Midas menu.

2. When the Run-Prog menu appears, select INITIAL.

When the command file is finished, the display will show some symbols of interest for the MemInit.Mc module. The various starting addresses are on pages 1 and 2 of Initial.Mc; they include the following:

SAAltoStart	boots and starts AMesa from the SA4000.
SAPilotStart	boots and starts Pilot from the SA4000.
EtherAltoStart	boots and starts AMesa from the 3 mb Ethernet.
EtherPilotStart	boots and starts Pilot from the 3 mb Ethernet.

Some interesting breakpoints in Initial are the following:

RamLoaded	(IMX 316) where LoadRAM finishes.
IMap	beginning of the map and storage test.
imRepeatStorageTest	after map test.; here you can change the contents of the SoftQThreshold register to 100b if you want to make the storage test put pages with correctable errors into service.
MemInitDone	end of storage test.
MicrocodeLoaded	after loading microcode into the VM block at 1400b.